SAPHIRE
Facets

SPONSORED BY MEMBERS OF THE SAPHIRE USERS GROUP

Volume 7 - Number 2
Fall 2002

Modeling Risk and Reliability

Contents

2 FAQ - Binning event tree sequences by initiators

4 Modeling System Upsets

8 Who's on First – Keeping Track of Basic Events

11 "Calc type" Corner - FALSE for Success

12 FAQ - Correlating basic events uncertainty

While the initial version of SAPHIRE (in the 1980s) was primarily designed as a teaching tool for reliability engineers (focusing on graphical fault trees) it has grown into a full-featured tool primarily for risk and reliability analysts. As such, an extensive list of features are available to assist in both the construction of a model and the subsequent analysis of the model. In this issue of the *Facets*, we cover topics ranging from system modeling techniques, methods for checking completed models, and ways in which to subdivide analysis results.

- Using fault trees to represent systems is a common practice, but in the “Modeling Systems Upsets” article we focus on potential modeling issues one must consider.

- In “Who’s on First...,” we illustrate the power behind the SAPHIRE relational database design by demonstrating the basic event cross-referencing feature where, at the touch of a button, one can find every place a basic event is used in a project.

- In the current installment of the “calc type corner,” we discuss the FALSE calculation type and provide information on how it is used both for basic event probabilities and when introduced into Boolean logic.

[illegible]

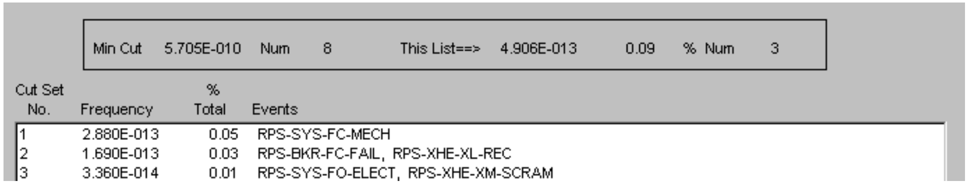
The SAPHIRE Facets is published by the Idaho National Environmental and Engineering Laboratory.
The editor is Dr. Curtis L. Smith and may be reached at P.O. Box 1625, Idaho Falls, ID 83415-3850 or by e-mail: CLS2@inel.gov

SAPHIRE Facets is published twice a year. Article submissions are welcome and can be sent to the editor.

FAQ - Is there a way to bin event tree sequences by initiators such that a numerical rank (by initiator) can be obtained from the results?

There are two ways to accomplish this feat, one using the cut set “slice” feature and the second using the end state partition rules. The “slice” feature is the easier of the two, but does not store the results as a permanent part of the database.

To use the "slice" feature, highlight all sequences of interest then select **Display ➜ Cut Sets**. Click the **Slice** button. You will then see a list of basic events – these are the events that show up in the sequence cut sets. Included in this list are all of the initiating events that appear in the sequences cut sets. Consequently, if you want to look at just the cut sets from the *IE-1* sequences, select the *IE-1* event in the list, click the "**-->**" button, and then click the **Apply** button. You would then see a window with just those cut sets that have *IE-1* in them. The top of the window has the total results (i.e., all cut sets) and the subset containing the *IE-1* cut sets in the sliced part (labeled "This List").



```
Min Cut    5.705E-010    Num    8        This List==>    4.906E-013    0.09    %  Num    3

Cut Set                    %
  No.     Frequency      Total    Events
1         2.880E-013     0.05     RPS-SYS-FC-MECH
2         1.690E-013     0.03     RPS-BKR-FC-FAIL, RPS-XHE-XL-REC
3         3.360E-014     0.01     RPS-SYS-FO-ELECT, RPS-XHE-XM-SCRAM
```

An alternative way to split the cut sets based upon the initiating event is to develop partition-based end state rules. Partition rules are a set of programmatic rules that subdivide sequence cut sets into user-defined end states. To enter partition rules, highlight the sequences of interest then select the **Cut Set ➜ Partition ➜ Edit Project** option. Then, in the rule editor, you would enter rules that look like:

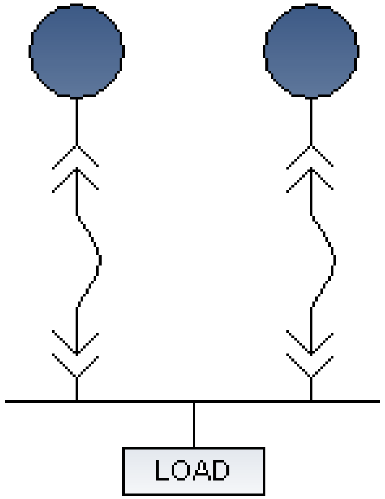
```
if  init(IE-1)  then
      partition = "ES-IE-1";
elsif  init(IE-2)  then
      partition = "ES-IE-2";
elsif  init(IE-3)  then
      partition = "ES-IE-3";
else
      partition = "ES-REMAINING-IES";
endif
```

After finishing and saving the rule, you would apply the partition rule via the **Cut Set ➜ Partition ➜ Apply Batch** option. Now, you would proceed to end states. Each end state partition specified in the rule above (ES-IE-1, ES-IE-2, etc.) will appear in the list of end states. Highlight these end states and select the **Gather** option. Specify that the cut sets are to be gathered by “Cut Set Partition” (this option tells SAPHIRE to use the partition rule above). After applying the gather option, SAPHIRE will gather all of the cut sets meeting the rule search criteria (e.g., cut sets containing IE-1) and put them into end state as specified (i.e., "ES-IE-1"). You can then view the cut sets for each end state or see the resulting total for each end state group. ❖

Modeling System Upsets

Curtis Smith

Representing system upset conditions via fault tree modeling can, in certain situations, be a challenge. For example, consider the simple power generation system (two generators supplying a single bus through two breakers) shown below. If the demand is such that any one generator can supply the load, one may immediately conclude that a fault tree with an AND gate could represent the system. Inputs into this AND gate would then be the generators and, perhaps, the circuit breakers.



One may then ask "what is the probability, within the next year, that I see an interruption of power to the load?" This question implies that I must assign applicable probabilities to the basic events feeding my AND gate described above. Since it was dictated that I am concerned about failures within a year, it may be natural to assign a failure probability (in SAPHIRE) using the nominal generator failure rate (assumed to be 1E-5 per hour) and a mission time of one year (8760 hours). Making this assignment in SAPHIRE, via the "calculation type" of "3" (operating component without repair, see *SAPHIRE Facets* Vol. 6 No. 2) yields a probability of:

P(generator) = 1 - exp(- rate * mission time)
= 1 - exp(- 1E-5 * 8760)
= 0.084

Now, since the generators feed into an AND gate, the system top event probability would be given as:

P(system) = P(generator A) * P(generator B) = 0.084 * 0.084 = 7.1E-3

The calculation above excludes the potential for common-cause failure. But, is the result of 7.1E-3 really what I was looking for from SAPHIRE in the first place? I had expressed that I would like to know the probability of an interruption of power. Unfortunately, what I have calculated is the probability that I see two events, generator A fails (and is not repaired) *and* generator B fails (and is not repaired). This calculation is not what I was looking for, since in the real world a failed generator would be repaired – most likely – as soon as possible. Thus, I have calculated a probability for an event that would not be allowed to happen.

Since I know that SAPHIRE has an unavailability model where the potential for repair is included, one may be tempted to utilize that calculation type to estimate the system interruption probability. Recall that calculation type "5" has, in addition to the failure rate and mission time, a parameter called the mean time to repair (see *SAPHIRE Facets* Vol. 7 No. 1). Let us assume that the mean time to repair for a generator is eight hours. I then find, using SAPHIRE, that the unavailability of a generator over a year, given that it can be repaired, is:

P(generator) = 8E-5

Again, I return to the fault tree (i.e., my AND gate) and find the system top event probability to be:

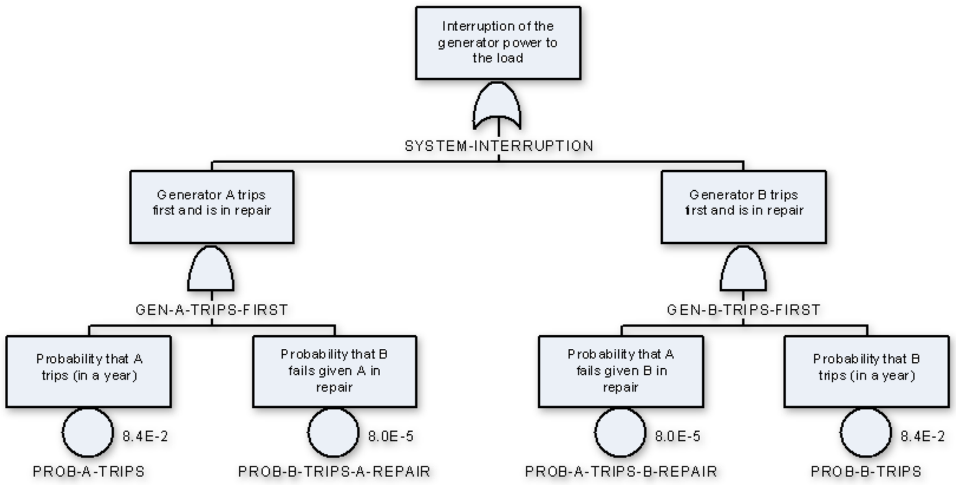
P(system) = P(generator A) * P(generator B) = 8E-5 * 8E-5 = 6.4E-9

Again, I question whether this result is really what I was looking for from SAPHIRE? Unfortunately, what I have calculated is the probability that two generator outages (a failure and then the time it takes to repair) *overlap* in time. Since the repair time is eight hours, this overlap of two failed generators would be within the eight hours. Since it is unlikely to have one generator out for eight hours (8E-5), it is even more unlikely to (independently) have two generators out at the same time. Hence, the low value of 6.4E-9. But, this probability – while interesting – is still not what I am looking for from SAPHIRE. Instead, what I want to know is the probability of seeing an interruption of generator power to the load.

As we can begin to see, the *context* of the system modeling not only dictates the structure of my fault tree model, but also plays an integral part of the probabilities that are introduced into the logic model. While the system hardware and operation does not change, my unreliability model "morphs" depending on the question I am asking. Now, let us finally answer the question "what is the probability, within the next year, that I see an interruption of power to the load?"

An interruption is defined as a loss of one generator followed by a subsequent loss of the second generator. Recall that earlier I indicated two important boundary conditions: (1) common-cause failures do not exist and (2) a generator can be repaired in eight hours (on average). Consequently, one should model failure of one generator followed by a subsequent failure of the second generator while the first is still being repaired. It is this scenario that will lead me to the interruption probability that I desire.

So, how do we best model this outage situation? Since the scenario is a temporal situation, it would be natural to turn to an event tree structure, where the initiating event is failure of one generator while the next top event is failure of the remaining generator during the repair time. Alternatively, one could represent the same scenario in a single fault tree. I chose this second option for the sake of simplicity, and our SAPHIRE fault tree is shown on the next page.



Interruption of the generator power to the load
SYSTEM-INTERRUPTION
Generator A trips first and is in repair
GEN-A-TRIPS-FIRST
Generator B trips first and is in repair
GEN-B-TRIPS-FIRST
Probability that A trips (in a year)
8.4E-2
PROB-A-TRIPS
Probability that B fails given A in repair
8.0E-5
PROB-B-TRIPS-A-REPAIR
Probability that A fails given B in repair
8.0E-5
PROB-A-TRIPS-B-REPAIR
Probability that B trips (in a year)
8.4E-2
PROB-B-TRIPS

The basic event probabilities are:

P(first generator fails)
= 1 - exp(-rate*mission time)
= 1 - exp(- 1E-5 * 8760)
= 0.084

P(second generator|other in repair)
= 1 - exp(- rate * repair time)
= 1 - exp(- 1E-5 * 8)
= 8.0E-5

From SAPHIRE, we find that the top event probability is 1.3E-5, which represents the probability of an interruption of power over the next year. Note though that even this calculation is (technically) not correct since I ignored the potential of seeing multiple generator trips (and subsequent repairs). To be precise, I should have represented situations where one generator trip occurs, two trips occur, three trips occur, etc. But, since the failure rate is low in this case, the probability of seeing two or more trips is about 0.004, which turns out to be approximately 5% of the probability of one trip. To close the discussion, it should be noted that the "correct" answer of 1.3E-5 is between the two extremes found earlier (6.4E-9, representing the eight hour outage, and 7.1E-3, representing two trip events). Consequently, our earlier crude modeling provided a lower and upper bound to the correct answer, but still represented a significant deviation from the correct results. ❖

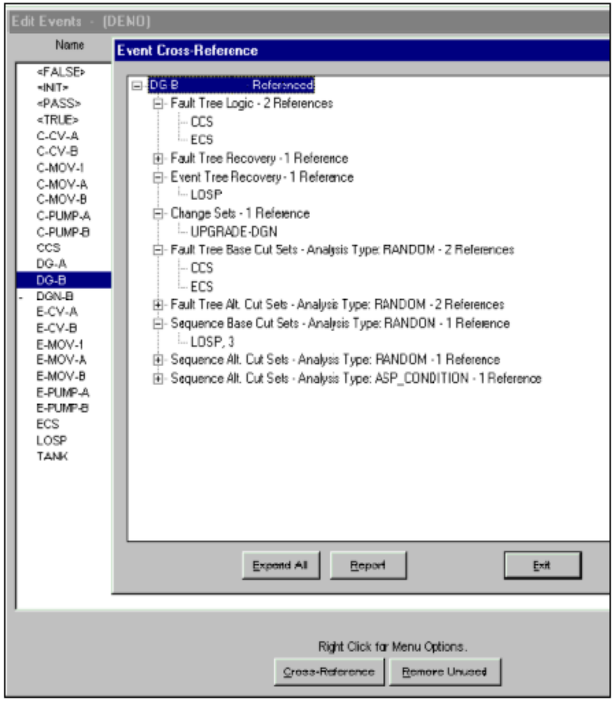
Who's on First – Keeping Track of Basic Events

Kellie Kvarfordt

The basic event is the core building block of any SAPHIRE risk or reliability model. Basic events are inherent to fault tree logic, cut sets, change sets, recovery rules, and more. But, since a basic event can be used in so many places in a project, it is easy to lose track of them. To build, validate, or just better understand a model, it is useful to locate the areas in a project that a particular event can influence. This article discusses how to track basic event usage through SAPHIRE's *cross referencing* features.

There are a couple of places in SAPHIRE where basic event usage can be reported. The first place is via the **Modify ➜ Basic Event** option. This option displays a master list of basic events for the current project. From here, select a basic event and press the **Cross Reference** button. SAPHIRE will create a hierarchical report containing a comprehensive listing of places that event is used.

Click the ⊞ (plus) and ⊟ (minus) boxes to expand and contract the list to focus on particular areas, or press the **Expand All** button to list the complete details for the basic event.

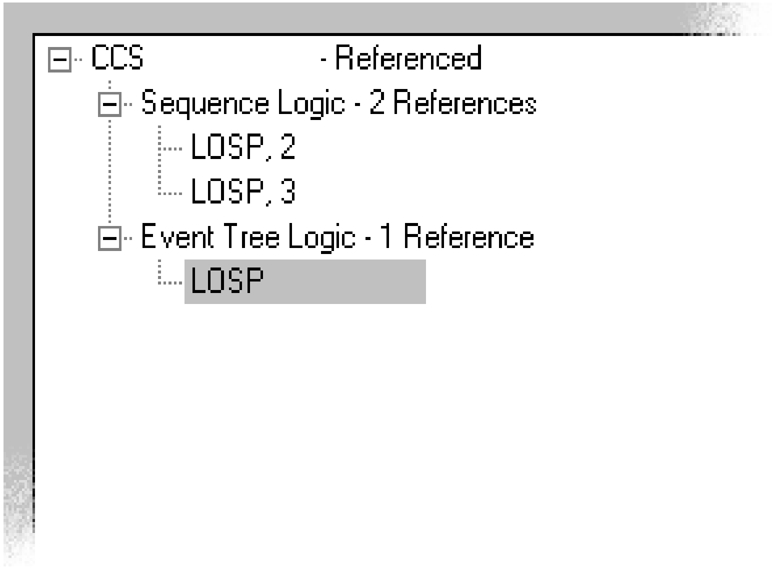


```
<FALSE>
<INIT>
<PASS>
<TRUE>
C-CV-A
C-CV-B
C-MOV-1
C-MOV-A
C-MOV-B
C-PUMP-A
C-PUMP-B
CCS
DG-A
DG-B
DGN-B
E-CV-A
E-CV-B
E-MOV-1
E-MOV-A
E-MOV-B
E-PUMP-A
E-PUMP-B
ECS
LOSP
TANK
```

```
⊟ DG B                Referenced
   ⊟ Fault Tree Logic - 2 References
        CCS
        ECS
   ⊞ Fault Tree Recovery - 1 Reference
   ⊟ Event Tree Recovery - 1 Reference
        LOSP
   ⊟ Change Sets - 1 Reference
        UPGRADE-DGN
   ⊟ Fault Tree Base Cut Sets - Analysis Type: RANDOM - 2 References
        CCS
        ECS
   ⊞ Fault Tree Alt. Cut Sets - Analysis Type: RANDOM - 2 References
   ⊟ Sequence Base Cut Sets - Analysis Type: RANDOM - 1 Reference
        LOSP, 3
   ⊞ Sequence Alt. Cut Sets - Analysis Type: RANDOM - 1 Reference
   ⊞ Sequence Alt. Cut Sets - Analysis Type: ASP_CONDITION - 1 Reference
```

Note that unused events appear in the Edit Events window marked with a dash (-) to the left of the event name. Cross reference reports for those events will be empty. All unused basic events can be deleted from the project at once by pressing the **Remove Unused** button, or individually by selecting the basic event, right clicking, and choosing the **Delete** menu option. To maintain data integrity, SAPHIRE will only remove those events marked as unused. To delete a referenced event, use the cross reference report as a handy way to locate and remove any undesired references (e.g., the event is used in a fault tree or recovery rule) to an event so that it can be removed from the project.

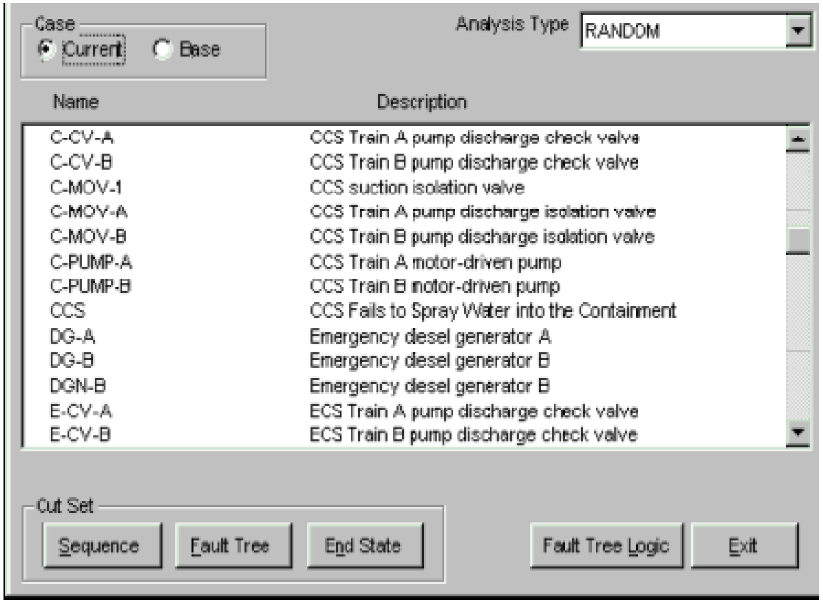
Note that SAPHIRE Version 7.x extends this "one-stop shopping" cross-reference feature to both fault trees and end states as well. For example, at right, we show how the CCS fault tree is used in the DEMO project in SAPHIRE 7.x. Here we note that the CCS fault tree is located in two sequences, LOSP 2 and LOSP 3, and can be found in the logic for the LOSP event tree.



```
CCS - Referenced
    Sequence Logic - 2 References
        LOSP, 2
        LOSP, 3
    Event Tree Logic - 1 Reference
        LOSP
```

Reporting Cross References

A second basic event cross reference option is available through the **Report ➜ Basic Event ➜ Cross Reference** menu option. From this option, SAPHIRE reports where selected basic events are found in either cut sets or fault tree logic, depending upon the report button selected (see next page).


Case
(● Current) ○ Base

Analysis Type: RANDOM

Name	Description
C-CV-A	CCS Train A pump discharge check valve
C-CV-B	CCS Train B pump discharge check valve
C-MOV-1	CCS suction isolation valve
C-MOV-A	CCS Train A pump discharge isolation valve
C-MOV-B	CCS Train B pump discharge isolation valve
C-PUMP-A	CCS Train A motor-driven pump
C-PUMP-B	CCS Train B motor-driven pump
CCS	CCS Fails to Spray Water into the Containment
DG-A	Emergency desel generator A
DG-B	Emergency desel generator B
DGN-B	Emergency desel generator B
E-CV-A	ECS Train A pump discharge check valve
E-CV-B	ECS Train B pump discharge check valve

The report cross reference feature is useful for summarizing the usage of multiple basic events in a single report. However, this option prints cross reference information only for one type of event usage per report. For example, the **Sequence** button shows only sequences which contain (via cut sets) the selected basic event(s). The **Fault Tree** button shows only fault trees which contain the selected basic event(s) while the **End State** button list the end states containing the selected basic events(s).

The most common uses of basic events – as a part of cut sets and logic – are available via these report options. More subtle usages, such as in change sets and recovery rules, are available only through the **Modify ➜ Basic Events ➜ Cross Reference** option discussed earlier.

One should note that if the results of the cross reference reports appear to contain erroneous information or omissions, SAPHIRE's cross reference maps may need to be updated. Choosing the **Utility ➜ Recover Data Base** menu option will update the database so that the report will display the correct information. ❖

"Calc type" Corner - FALSE for Success

Curtis Smith

If one looks at the basic event data options (e.g., via **Modify ➜ Basic Event**), under the "Random Failure Data" category of information is a field called "type." Normally, it seems that this information is used to indicate a probability calculation for the basic event. But, the calculation type, or "calc type" in SAPHIRE lingo, is used for more than just denoting a probability equation. As we have discussed in previous *SAPHIRE Facets*, calc types have a variety of uses, just one of which is to indicate a probability calculation equation for a basic event. In this issue, we will discuss calculation type **F** or FALSE.

Calculation Type F

The **F** calculation type represents a logically FALSE house event. In general, Boolean house events have either probability one (if TRUE) or zero (if FALSE). Since SAPHIRE fault trees are based on the concept of failure space (i.e., the basic event probabilities nominally represent the likelihood of failing), having a probability of zero implies that the basic event is not failed. In other words, a FALSE house event signifies a success event. A basic event that has been set to **F** has a nominal probability of zero. In addition though, SAPHIRE treats these events in a special manner in order to determine the correct minimal cut set when evaluating the Boolean logic indicated via the fault or event tree. Specifically, when the logic is solved, SAPHIRE prunes the logic depending on the structure of the tree and type of house event. Depending on the gate type that contains the **F** event, SAPHIRE will:

If the gate is an AND gate, then the *gate* is set to **F**.
If the gate is an OR gate, then the **F** event is removed from the gate input list.

Of course, since events feed into gates which (perhaps) feed into other gates, SAPHIRE will propagate the **F** event up through the tree when applicable. For example, if basic event *ABC* (set to **F** calc type) is input into an AND gate *G1*, then *G1* becomes an event (like *ABC*) and is set to **F**. If gate *G1* is an input to gate *G2* and *G2* is an AND gate, then *G2* will be turned into an event and set to **F**. This process is repeated until (1) the top of the tree is reached or (2) the **F** event reaches an OR gate. If the top of the tree is reached and the top gate is an AND gate, then the tree has no minimal cut sets. In this case, SAPHIRE will display the message "The TOP event cannot occur (FALSE)!"

Conclusions

For calc type **F**, SAPHIRE offers the user two primary uses. First, it is used if a component is deemed to be perfect (failure probability equals zero). Second, it is used to “turn off” portions of the fault tree logic, thereby adjusting the fault tree without having to create an entirely new tree. ❖

FAQ - How do I correlate individual basic events during uncertainty sampling?

If basic events are left uncorrelated, then they are treated as being independent from one another during the uncertainty sampling. To correlate basic events, first make sure that each basic event in the correlation “group” has the same mean value and uncertainty distribution. Then, select each basic event via the **Modify ➜ Basic event** option. For each event, enter a correlation class identifier. Note that the identifier must be the same for each event in the correlation group and consists of one to four letters and/or numbers. ❖